Chapter 4:

Gate Level Minimization

Introduction

- Complexity of digital circuit & complexity of algebraic expression.
  - A function's truth-table representation is unique; its algebraic expression is not.
  - Simplification by algebraic means is awkward (from algorithmic point of view);
  - Algebraic manipulations is hard since there is not a uniform way of doing it

- The map method called Karnaugh map (or K-map) [M. Karnaugh, 1953] is straightforward and commonly used.

K-map

- A diagram made up of squares each representing one minterm.
  - A pictorial form of the truth table;
  - A visual diagram of all possible ways a function may be expressed—the simplest one can easily be identified.
- It produces a circuit diagram with a minimum number of gates and the minimum number of inputs to the gate.
- It is sometimes possible to find two or more expressions that satisfy the minimization criteria.

Logic circuit $\Leftrightarrow$ Boolean function $\Leftrightarrow$ Truth table $\Leftrightarrow$ K-map

$\Leftrightarrow$ Canonical form (sum of minterms, product of maxterms)

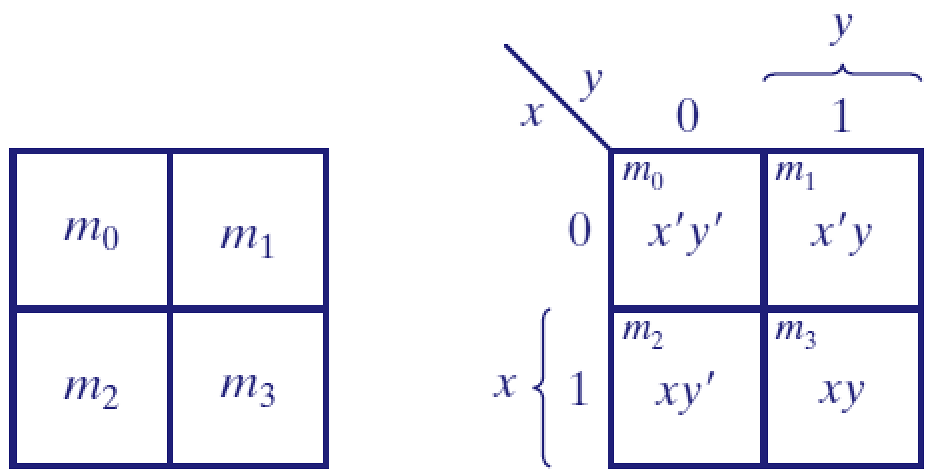
$\Leftrightarrow$ (Simplifier) standard form (sum of products, product of sums)

Two-Variable K-Map

- Two-variable has four minterms, and consists of four squares.

$$m_1 + m_2 + m_3 = x'y + xy' + xy = x + y$$

- ✓ Four Minterms
- ✓ Two variables
- ✓ Four squares for four minterms



(a) (b)

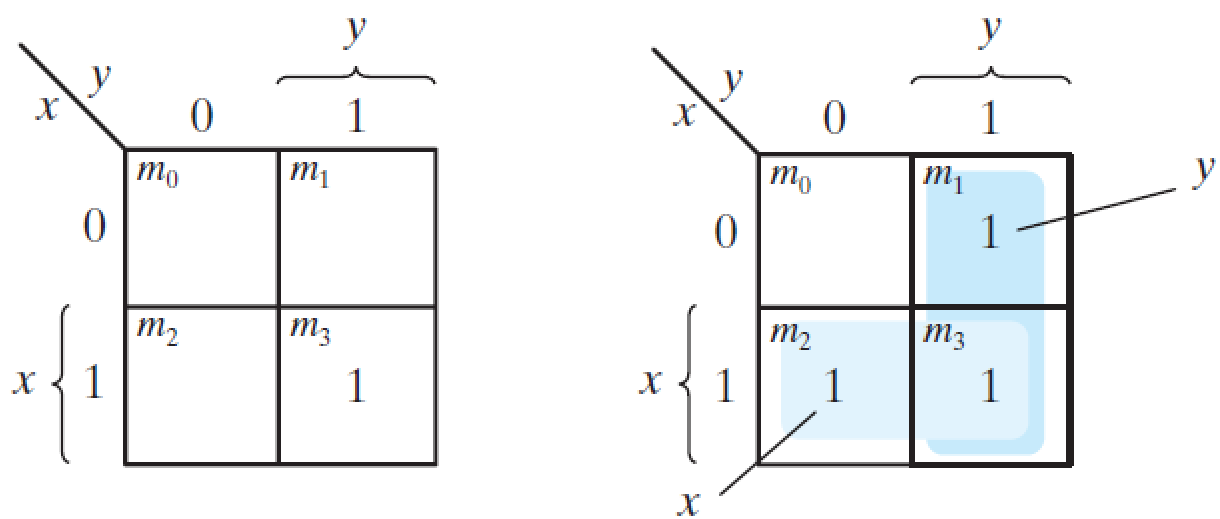
✓ Useful for representation of 16 Boolean functions.

Example:

i. If xy is equal to m_3, a 1 is placed inside the square that belongs to m_3.

ii. If $m_1 = m_2 = m_3 = 1$ *then*

$$m_1 + m_2 + m_3 = x'y + xy' + xy = x + y \text{ (OR function)}$$



(a) xy (b) $x + y$

Three-Variable K-Map

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

(a)

	yz: 00	01	11 (y)	10 (y)
x = 0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
x = 1	m_4 $xy'z'$	m_5 $xy'z$ (z)	m_7 xyz (z)	m_6 xyz'

(b)

- ✓ There are 8 minterms for 3 variables.
- ✓ So, there are 8 squares.
- ✓ Minterms are arranged not in a binary sequence, but in sequence similar to the Gray code.

Cont…

- Two adjacent squares differs by one variable (one primed other is not).
  - So, they can be minimized easily

Example:

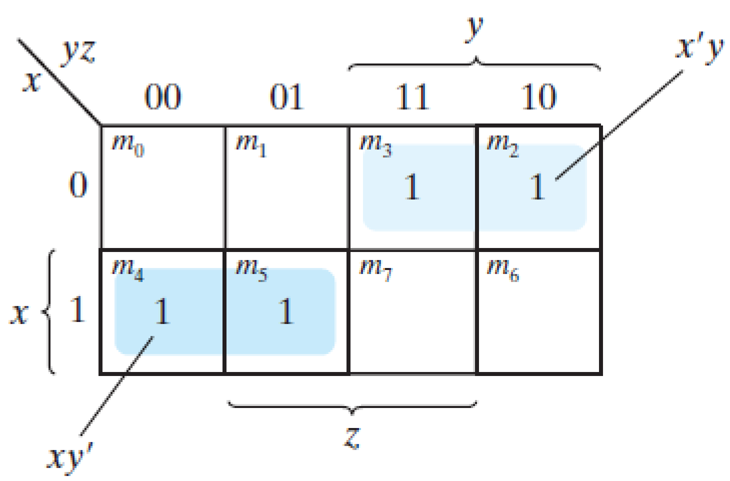
$$m_5 + m_7 = xy'z + xyz = xz(y' + y) = xz$$

Any two minterms in adjacent squares (vertically or horizontally, but not diagonally, adjacent) that are ORed together will cause a removal of the dissimilar variable.

Example

- Simplify the Boolean function

$$F(x, y, z) = \Sigma m(2, 3, 4, 5)$$



$$F(x,y,z) = x'y + xy'$$

Steps

i. a 1 is marked in each minterm square that represents the function.

ii. Find possible adjacent squares (Two shaded rectangles, each enclosing two 1's)

iii. The sum of four minterms can be replaced by a sum of only two product terms.

Cont…

- In some cases, some adjacent squares don't touch each other.

- E.g
  - ***m_0 is adjacent to m_2 and m_4 is adjacent to m_6.***

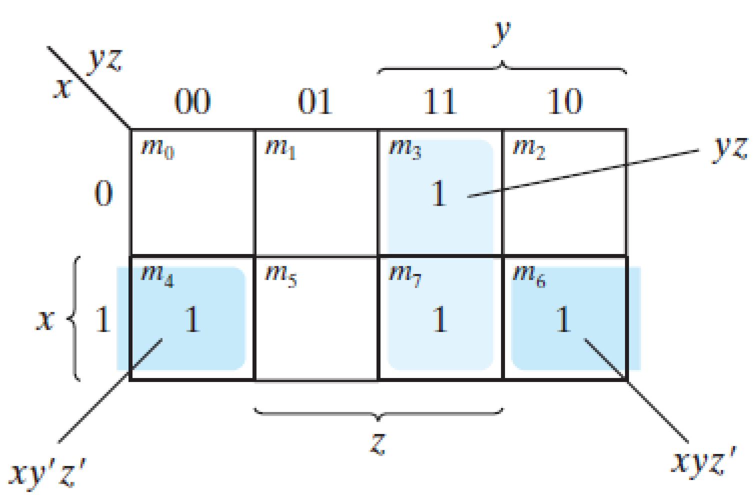
i. $m_0 + m_2 = x'y'z' + x'yz' = x'z'(y' + y) = x'z'$

ii. $m_4 + m_6 = xy'z' + xyz' = xz'(y' + y) = xz'$

Example

- Simplify the Boolean function

$$F(x,y,z)=\Sigma m(3,4,6,7)$$



$$F(x,y,z) = yz + xz'$$

Example

- A combination of four adjacent squares in the three-variable map represents the logical sum of four minterms and results in an expression with only one literal.

$$
\begin{aligned}
m_0 + m_2 + m_4 + m_6 &= x'y'z' + x'yz' + xy'z' + xyz' \\
&= x'z'(y' + y) + xz'(y' + y) \\
&= x'z' + xz' \\
&= z'(x' + x) \\
&= z'
\end{aligned}
$$

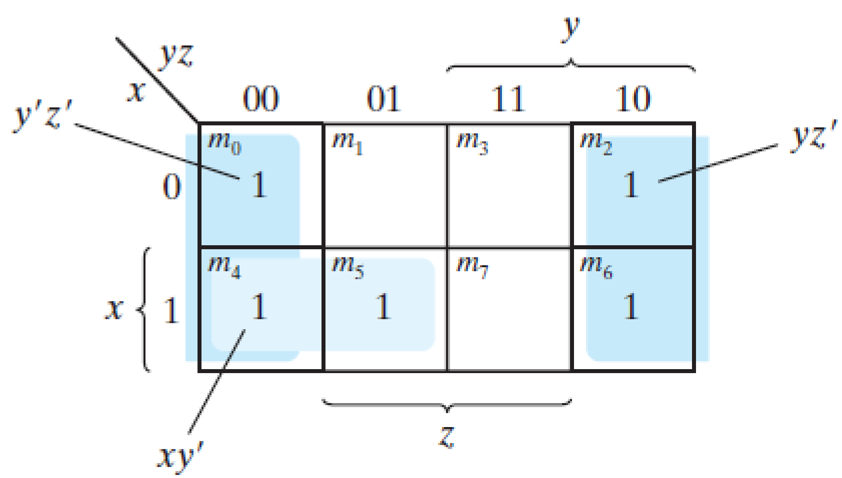
Example

- Simplify the Boolean function

$$F(x,y,z) = \Sigma m(0,2,4,5,6)$$

Sol

$$F(x,y,z)=z'+xy'$$



Summary

- On a 3-variable K-Map:
  - One square represents a minterm with three variables
  - Two adjacent squares represent a product term with two variables
  - Four “adjacent” squares represent a product term with one variable
  - Eight “adjacent” squares is the function of all ones (logic 1).

- In general,
  - The more the squares combined, the fewer the literals of a product term
  - Overlap is allowed.

Example

$F = AB'C' + AB'C + ABC + ABC' + A'B'C + A'BC'$

$A \backslash BC$	00	01	11	10
0	0	1	0	1
1	1	1	1	1

$F = A + B'C + BC'$

Example

- $F(x, y, z) = \Sigma m(2, 3, 6, 7)$

X \ YZ	00	01	11	10
0			**1**	**1**
X 1			**1**	**1**

Y (columns 11, 10); Z (columns 01, 11)

- Applying the Minimization Theorem three times:

$$F(x, y, z) = \bar{x}\, y\, z + x\, y\, z + \bar{x}\, y\, \bar{z} + x\, y\, \bar{z}$$

$$= yz + y\bar{z}$$

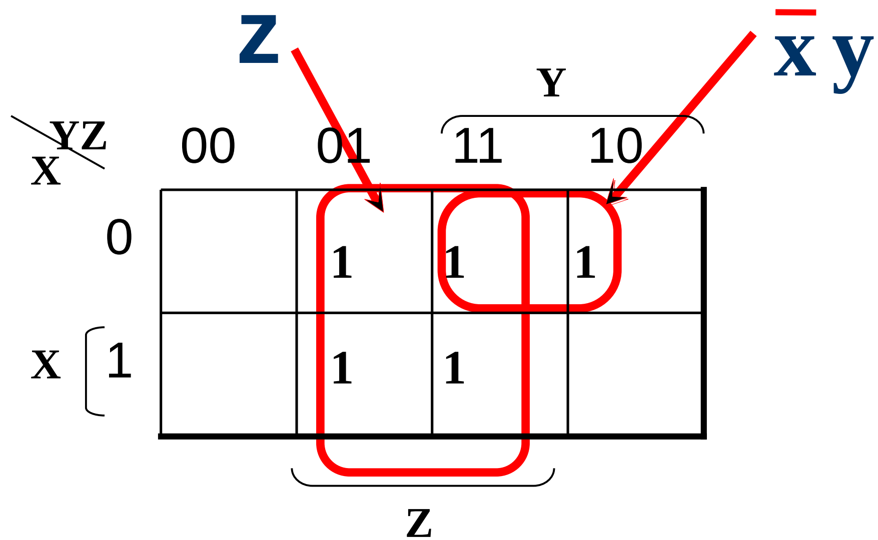
$$= y$$

Example:

- Simplify

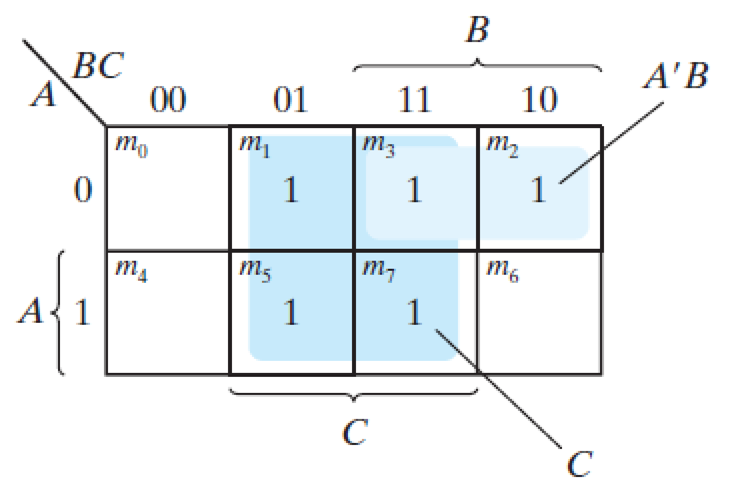
  F(X, Y, Z) = X'Z + X'Y + XY'Z + YZ

  F(X, Y, Z) = Σm (1, 2, 3, 5, 7)



Example

- $F = A'C + A'B + AB'C + BC$ *(Boolean function)*
  i. Express F as a sum of minterms.
  ii. Find the minimal sum-of-products expression



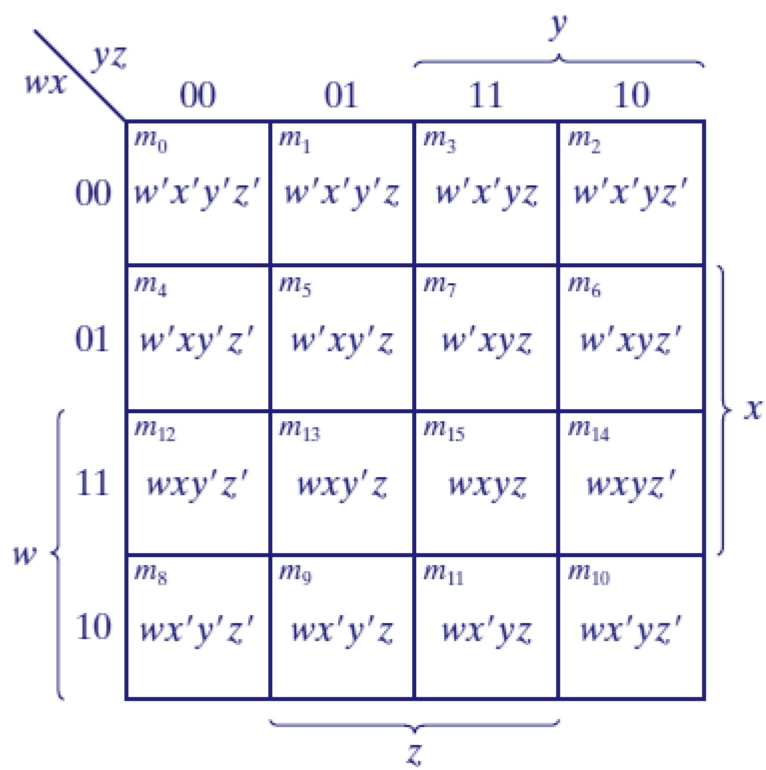
- *Sol*
  i. $F(A,B,C) = \Sigma m(1,2,3,5,7)$
  ii. $F = C + A'B$

Four-Variable Map

- 16 minterms (and squares) for 4 variables.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

(a)

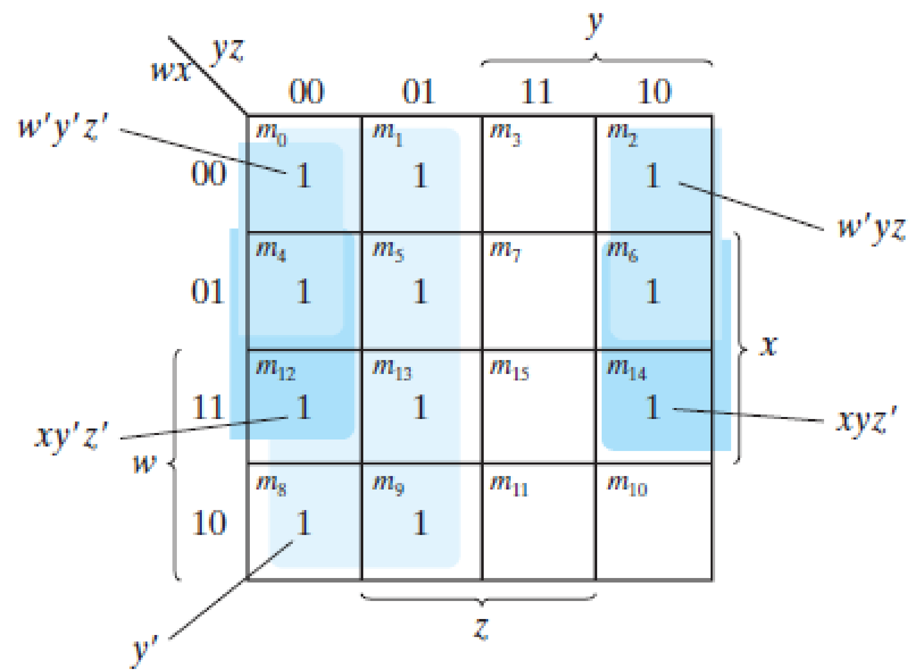


(b)

- ✓ One square represents one minterm, giving a term with four literals.
- ✓ Two adjacent squares represent a term with three literals.
- ✓ Four adjacent squares represent a term with two literals.
- ✓ Eight adjacent squares represent a term with one literal.
- ✓ Sixteen adjacent squares produce a function that is always equal to 1.

Example

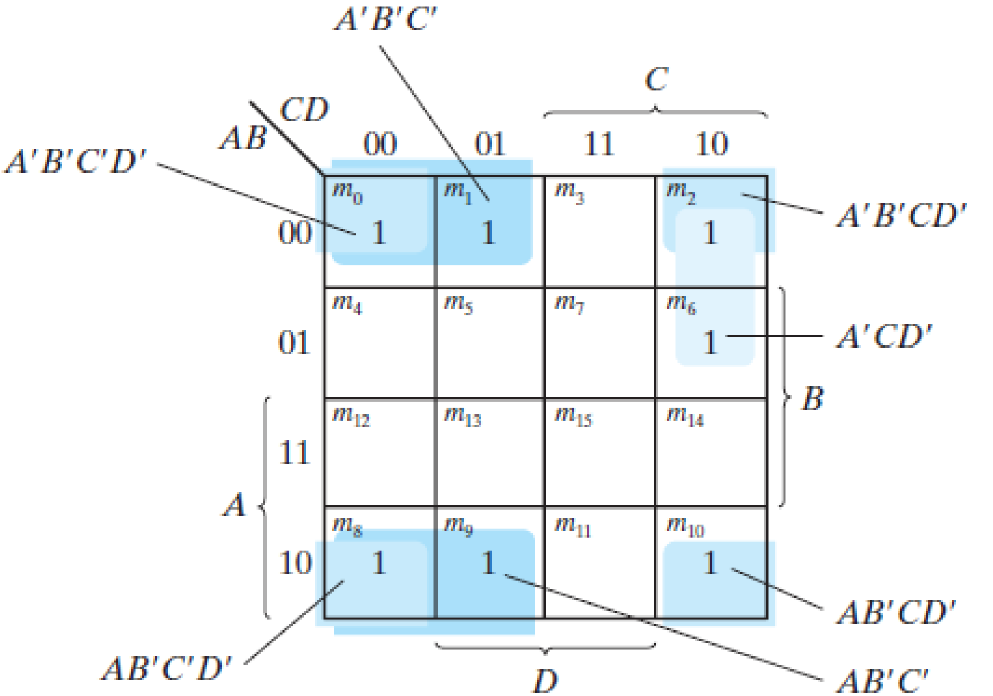
- Simplify $F(w,x,y,z) = \Sigma m(0,1,2,4,5,6,8,9,12,13,14)$



$F(w,x,y,z) = y' + w'z' + xz'$

Example

- Simplify $F = A'B'C' + B'CD' + A'BCD' + AB'C'$



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$
$A'B'C' + AB'C' = B'C'$

$F = B'D' + B'C' + A'CD'$

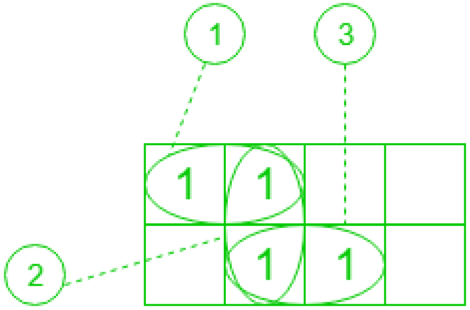
Prime Implicants

- A prime implicant is a product term obtained by combining the maximum possible number of adjacent squares in the map.

- If a minterm is covered by only one prime implicant, that prime implicant is said to be *essential*.

Prime Implicants

- This is a group of square or rectangle made up of bunch of adjacent minterms which is allowed by definition of K-Map. i.e. all possible groups formed in K-Map.

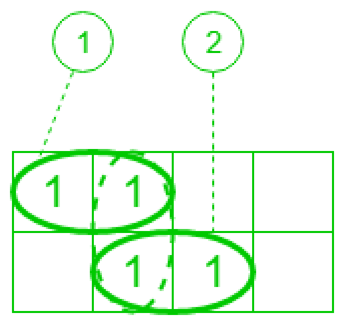


No. of Prime Implicants = 3

Essential Prime Implicants

- These are those subcubes (groups) which cover at least one minterm that can't be covered by any other prime implicant.

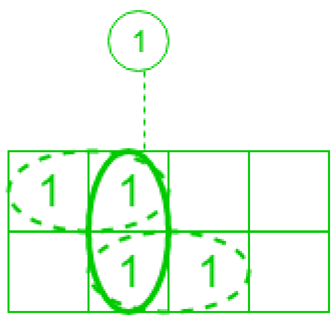
- Essential prime implicants(EPI) are those prime implicants which always appear in final solution.



No. of Essential Prime Implicants = 2

Redundant Prime Implicants

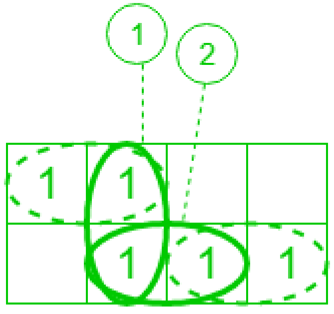
- The prime implicants for which each of its minterm is covered by some essential prime implicant are redundant prime implicants(RPI).
- This prime implicant never appears in final solution.



No. of Redundant Prime Implicants = 1

Selective Prime Implicants (SPI)

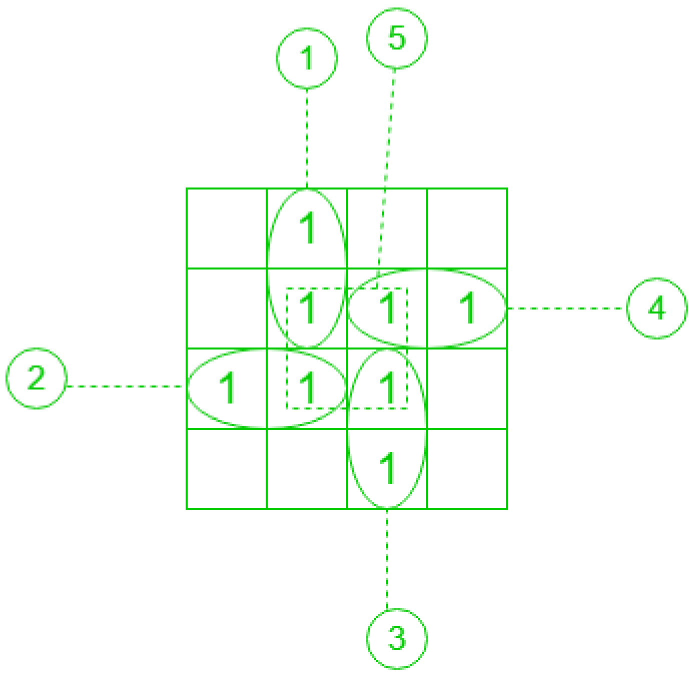
- These are the prime implicants for which are neither essential nor redundant prime implicants.
- These are also known as non-essential prime implicants.
- They may appear in some solution or may not appear in some solution.



No. of Selective Prime Implicants = 2

Example

- Given $F = \sum(1, 5, 6, 7, 11, 12, 13, 15)$, find number of implicant, PI, EPI, RPI and SPI.



No. of Implicants = 8

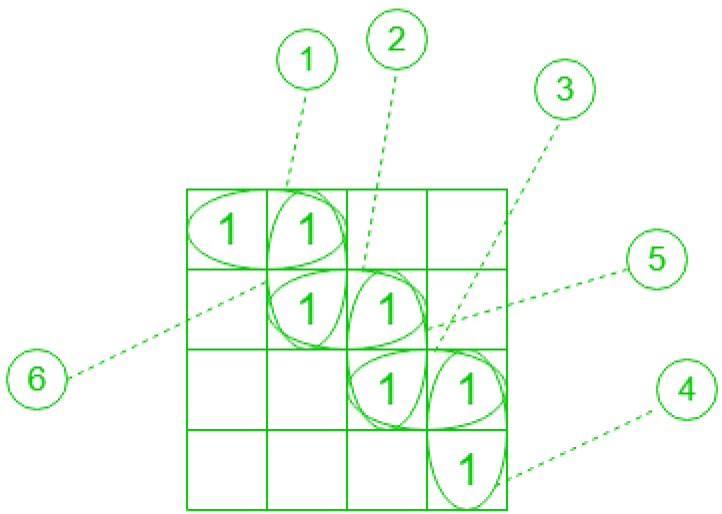
PI = (1,2,3,4,5)

EPI = (1,2,3,4)

RPI = (5)

F = 1 + 2 + 3 + 4

- Given $F = \sum(0, 1, 5, 7, 15, 14, 10)$, find number of implicant, PI, EPI, RPI and SPI.



No. of Implicants = 7

PI = (1,2,3,4,5,6)

EPI = (1,4)

SPI = (2,3,5,6)

F = 1 + 2 + 3 + 4

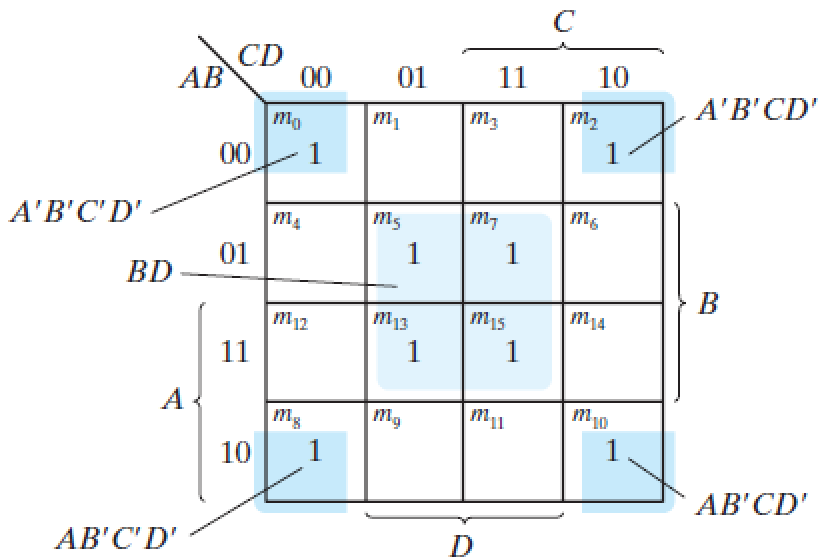
OR

F = 1 + 5 + 6 + 4

Example

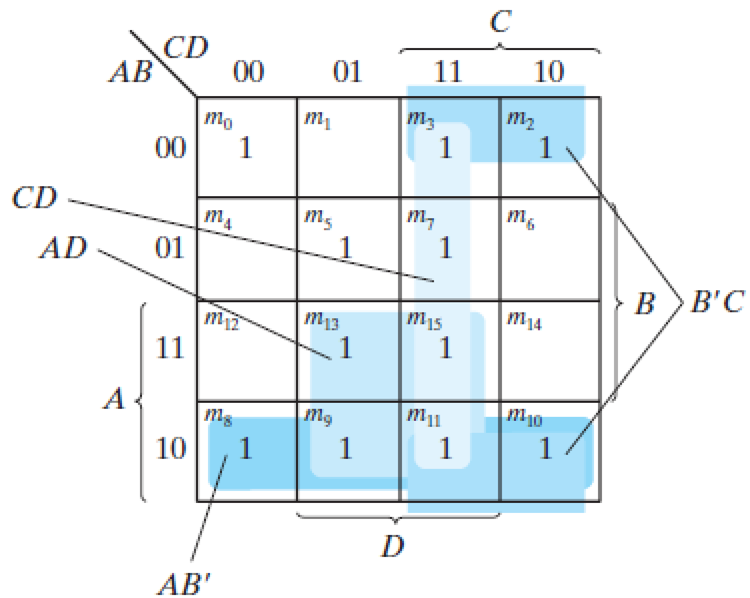
$$F = BD+B'D'+CD+AD$$
$$=BD+B'D'+CD+AB'$$
$$=BD+B'D'+B'C+AD$$
$$=BD+B'D'+B'C+AB'$$

- $F(A,B,C,D)=\Sigma m(0,2,3,5,7,8,9,10,11,13,15)$



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$

(a) Essential prime implicants BD and $B'D'$



(b) Prime implicants CD, $B'C$, AD, and AB'

Five-Variable Map

- Not simple, is not usually used.
  - 5 variables = 32 squares.
  - 6 variables = 64 squares.

- The left-hand four-variable map represents the 16 squares where A=0, and the other four-variable map represents the squares where A=1.

- In addition, each square in the A=0 map is adjacent to the corresponding square in the A=1 map.

$A = 0$

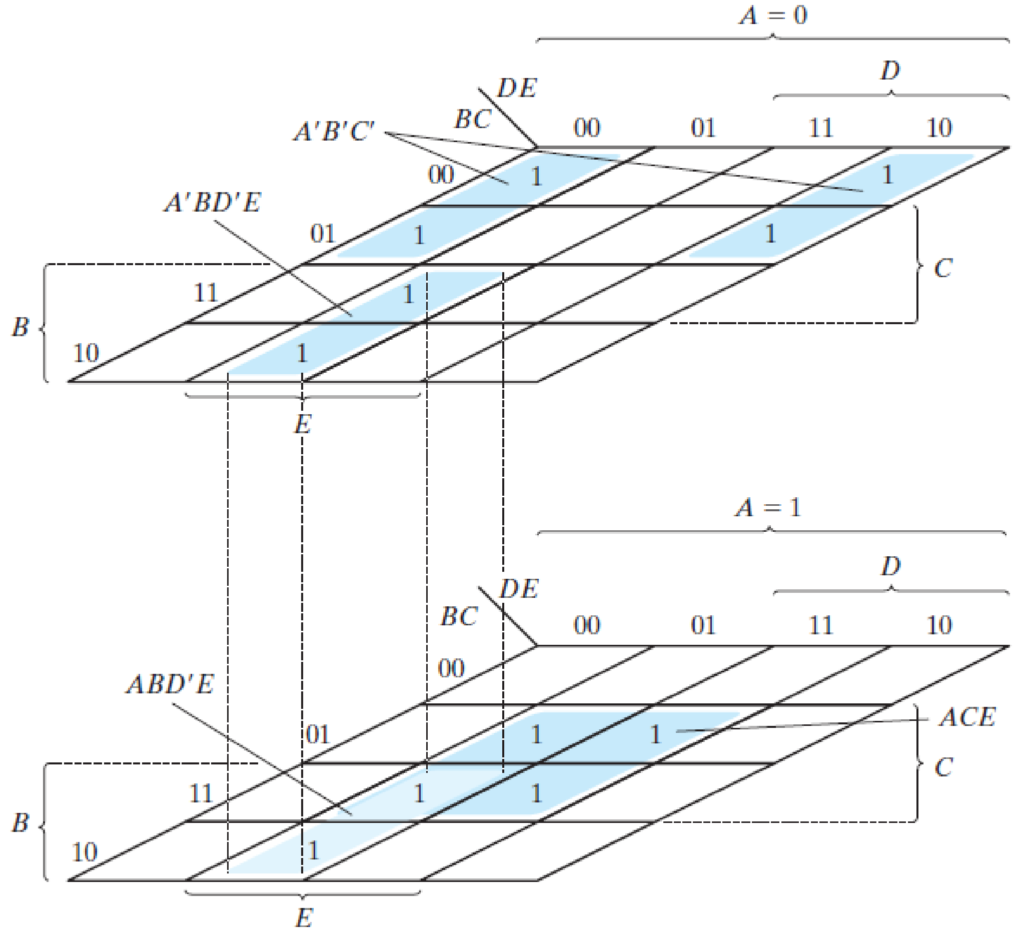
BC \ DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

$A = 1$

BC \ DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

- Simplify $F(A,B,C,D,E)=\Sigma m(0,2,4,6,9,13,21,23,25,29,31)$

$F = A'B'E' + BD'E' + ACE$



The Relationship Between the Number of Adjacent Squares and the Number of Literals In the Term

	Number of Adjacent Squares	Number of Literals in a Term in an *n*-variable Map			
K	2^k	$n = 2$	$n = 3$	$n = 4$	$n = 5$
0	1	2	3	4	5
1	2	1	2	3	4
2	4	0	1	2	3
3	8		0	1	2
4	16			0	1
5	32				0

Product-of-Sums Simplification

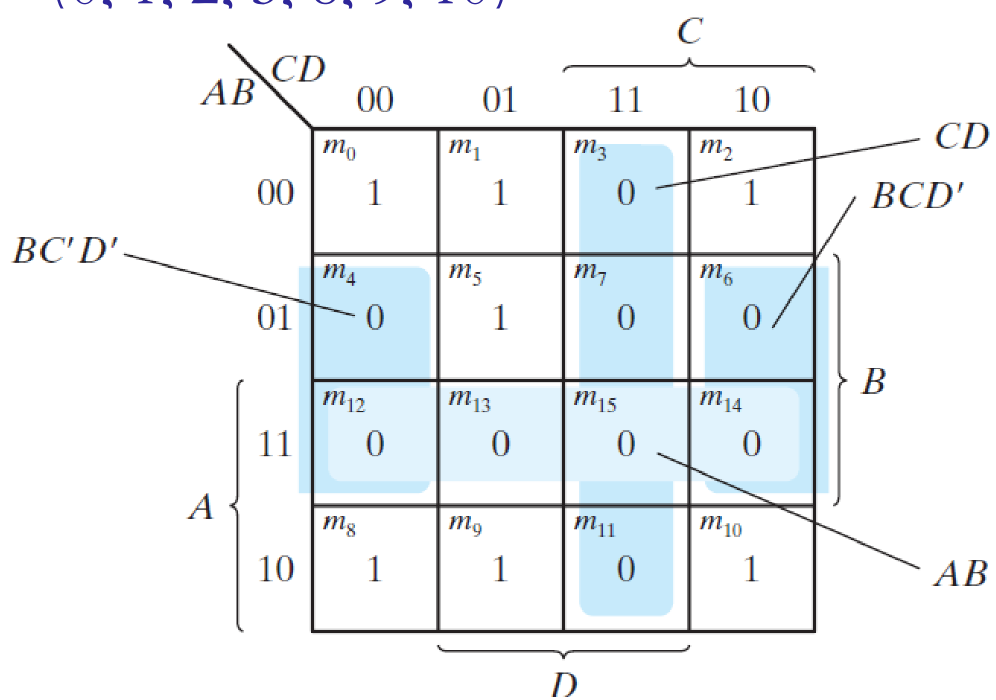
- Minterms represented by 1's placed in the squares of the map.
- The minterms not included is denote by its complement 0's.
- The complement of SOP of the function *F* gives us back the function *F* in POS form (a consequence of DeMorgan's theorem).

Example

- Simplify the following Boolean function into
  1. Sum-of-products form and
  2. Product-of-sums form:

$$F(A, B, C, D) = (0, 1, 2, 5, 8, 9, 10)$$

- Solution



- The 1's in the map represent all the minterms of the function.
- The squares marked with 0's represent the minterms not included in F and therefore denote the complement of F

1. SOP

- Combine the squares with 1's in the map

$$F = B'D' + B'C' + A'C'D$$

2. POS

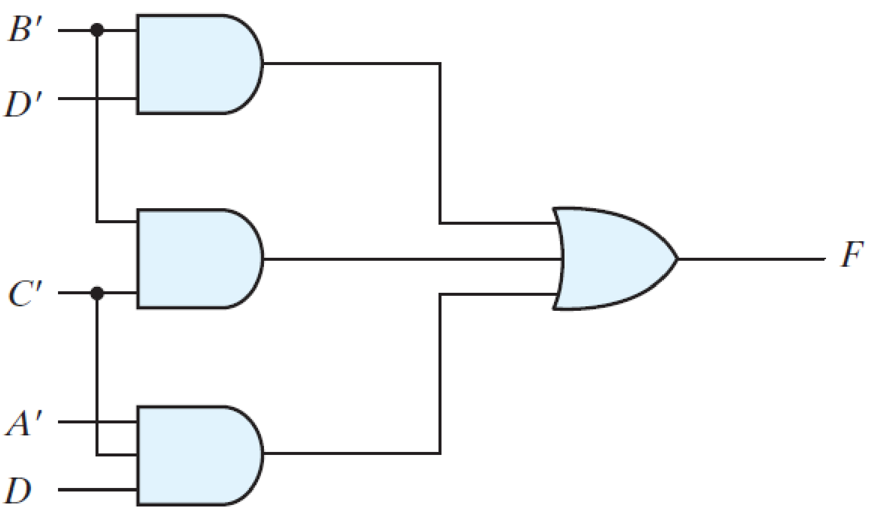
- Combining squares marked with 0's, we obtain the simplified complemented function:

$$F' = AB + CD + BD'$$

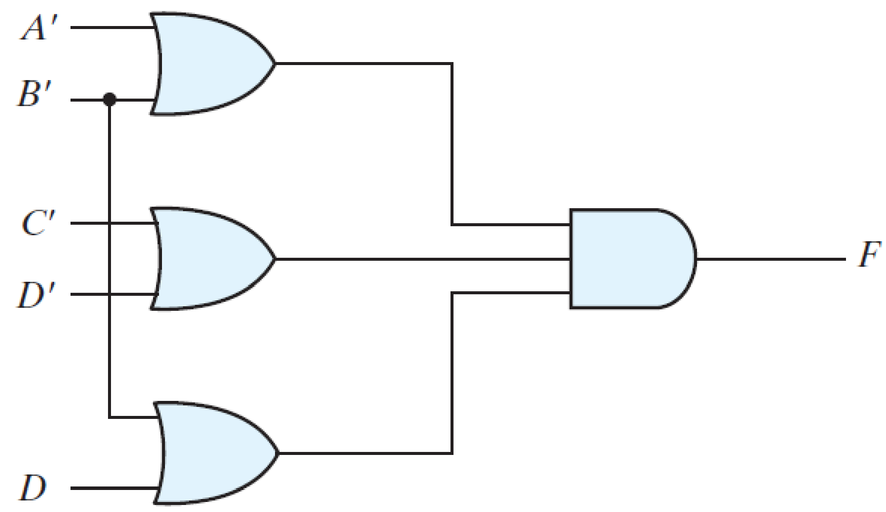
- Applying DeMorgan's theorem (to above function)

$$F = (A' + B')\,(C' + D')\,(B' + D)$$

$$F(A, B, C, D) = (0,1, 2, 5, 8, 9,10)$$
$$= B'D' + B'C' + A'C'D$$
$$= (A' + B')(C' + D')(B' + D)$$



(a) $F = B'D' + B'C' + A'C'D$



(b) $F = (A' + B')(C' + D')(B' + D)$

Example

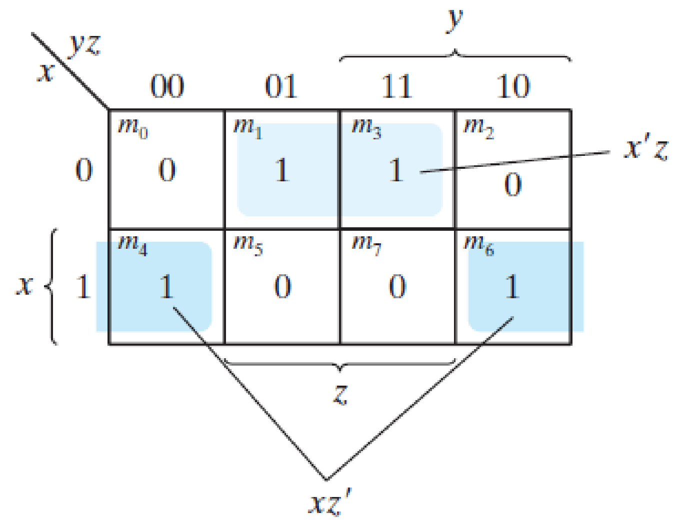
Express the following sum-of-minterms as POS

$$F(x, y, z) = \sum m(1, 3, 4, 6)$$

Sol

- In product-of-maxterms form

$$F(x, y, z) = \prod M(0, 2, 5, 7)$$



SOP

$$F = x'z + xz'$$

POS

- Combine the 0's and take the complement

$$F' = xz + x'z' \qquad F = (x' + z')(x + z)$$

Don’t-Care Conditions

- A combination of variables whose logical value is not specified.
  - Cannot be marked with a 1 or 0 in the map
  - Marked with an X

- An X inside a square in the map indicates that we don’t care whether the value of 0 or 1 is assigned to F for the particular minterm.

Example

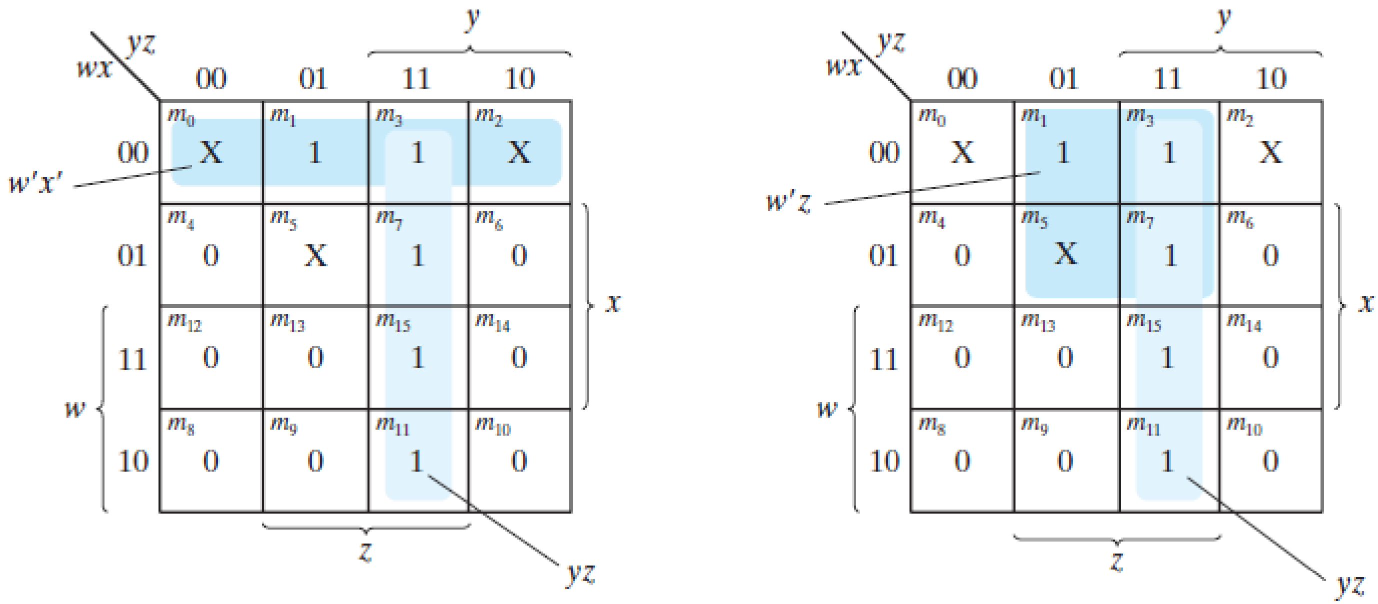
Simplify the Boolean function

$$F(w, x, y, z) = (1, 3, 7, 11, 15)$$

which has the don't-care conditions

$$d(w, x, y, z) = (0, 2, 5)$$

Sol



(a) $F = yz + w'x'$ (b) $F = yz + w'z$

- In (a),

$$F = yz + w'x'$$

$$F\,(w, x, y, z) = yz + w'x' = (0, 1, 2, 3, 7, 11, 15)$$

- In (b)

$$F = yz + w'z$$

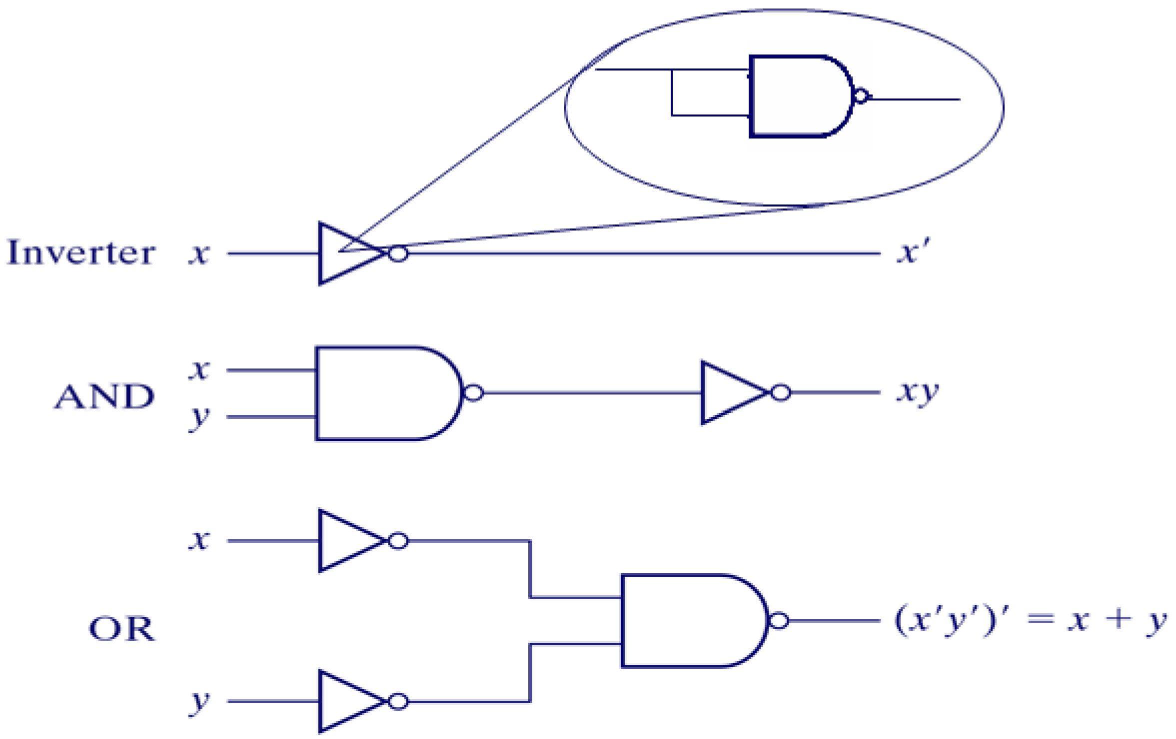
$$F\,(w, x, y, z) = yz + w'z = (1, 3, 5, 7, 11, 15)$$

NAND AND NOR Implementation

- Digital circuits are constructed with NAND or NOR
  - Easier to fabricate with electronic components
  - The basic gates used in all IC digital logic families.

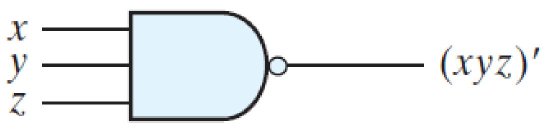
NAND Circuits

- Universal gate – any logic circuit can be implemented with it.

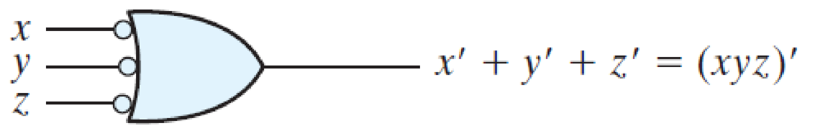


Logic Operations with NAND Gates

- Alternative graphic symbol for the NAND gate



(a) AND-invert



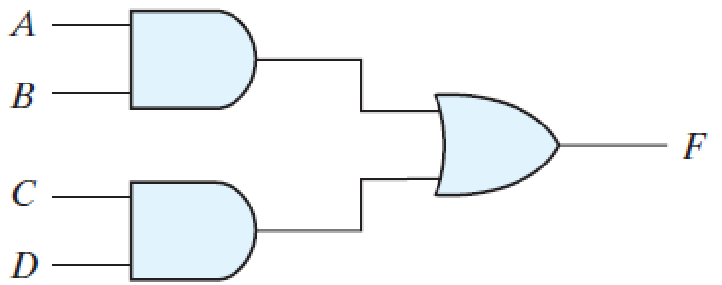
(b) Invert-OR

- Can be implemented in
  - 2 Level
  - Multilevel

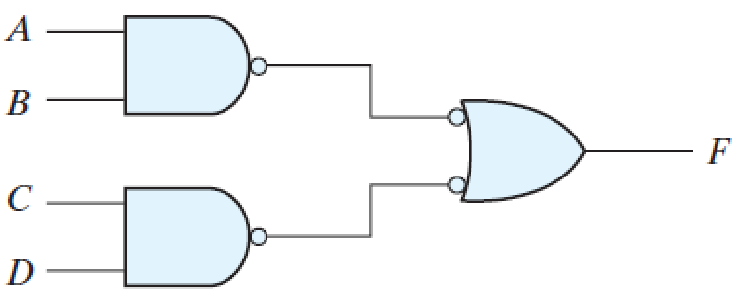
Two-Level Implementation

- Expressed in sum-of-products form.

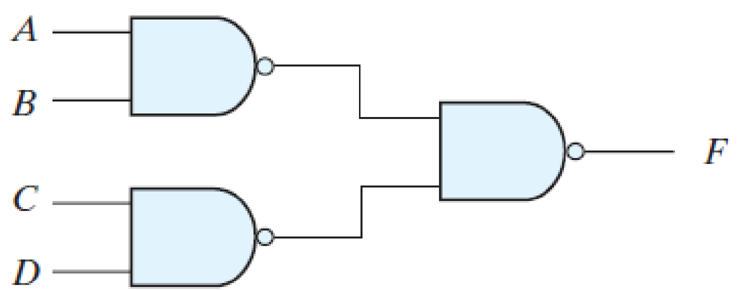
$$F = AB + CD = ((AB)'(CD)')'$$



(a)



(b)



(c)

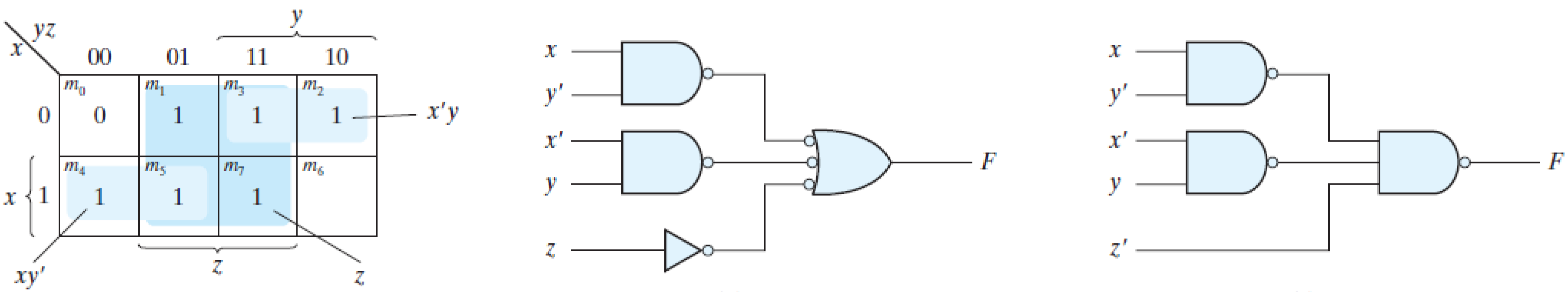
Example

- Implement the following Boolean function with NAND gates:

$$F(x, y, z) = (1, 2, 3, 4, 5, 7)$$

- Sol

$$F = xy' + x'y + z$$



(a) (b) (c)

Steps

1. Simplify the function and express it in SOP

2. Draw a NAND gate for each product term of the expression that has at least two literals. The inputs to each NAND gate are the literals of the term.

3. Draw a single gate using the AND-invert or the invert-OR graphic symbol in the second level, with inputs coming from outputs of first-level gates.

4. A term with a single literal requires an inverter in the first level. However, if the single literal is complemented, it can be connected directly to an input of the second level NAND gate.

Multilevel NAND Circuits

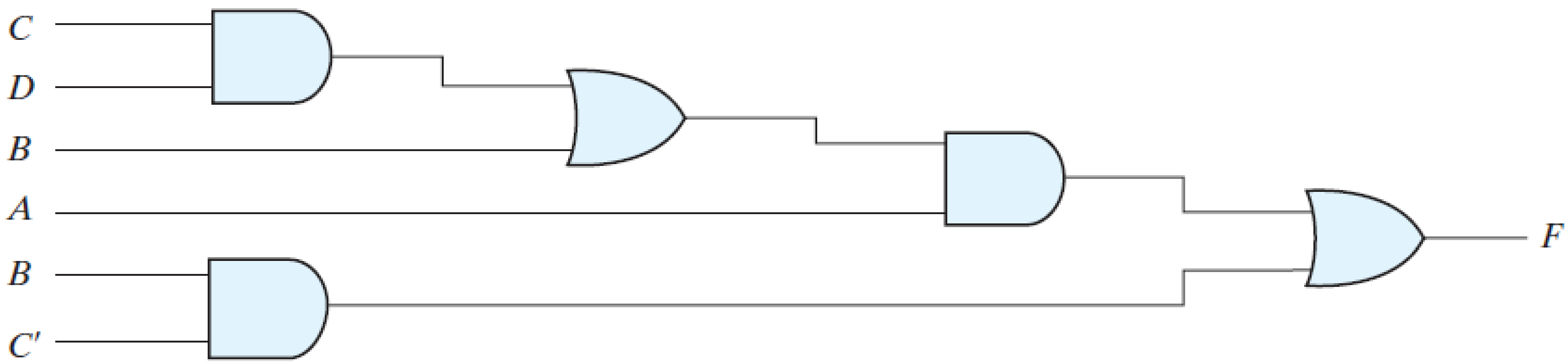
- Consider the Boolean function

$$F = A\,(CD + B) + BC'$$

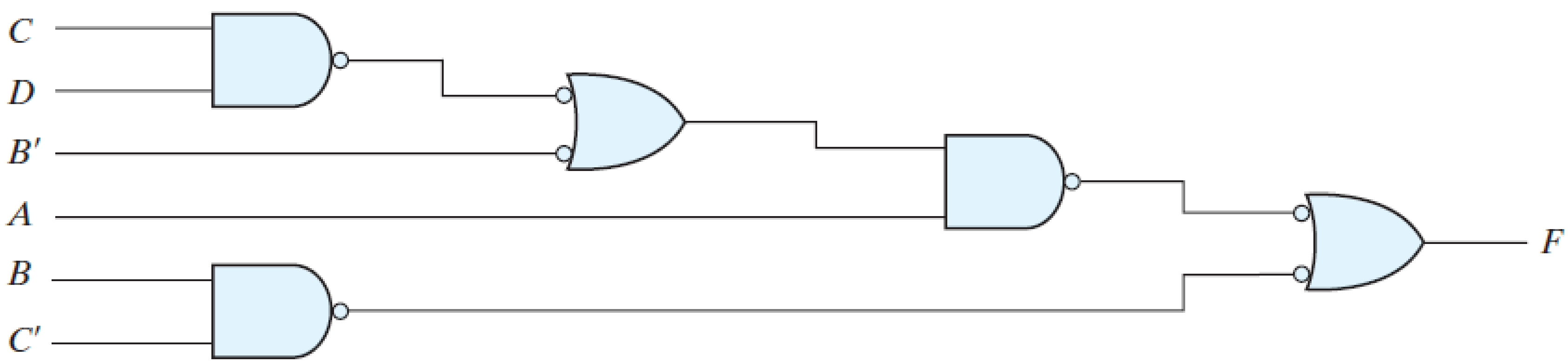
Sol

- Steps

1. Convert all AND gates to NAND gates with AND-invert graphic symbols.
2. Convert all OR gates to NAND gates with invert-OR graphic symbols.
3. Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (a one-input NAND gate) or complement the input literal.



(a) AND–OR gates



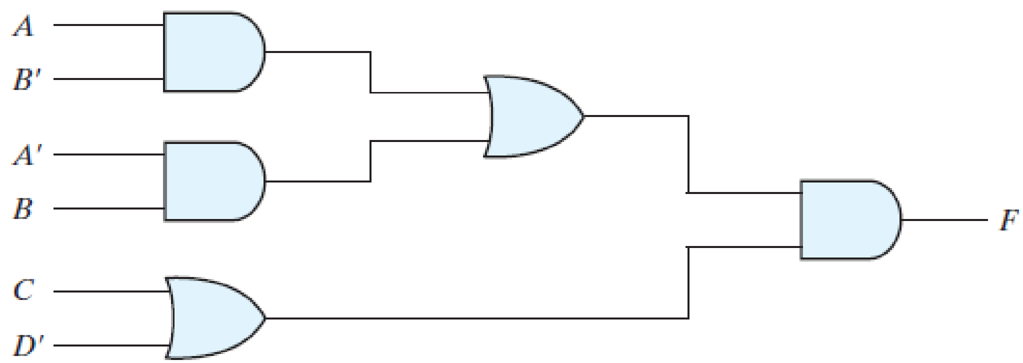
(b) NAND gates

Example

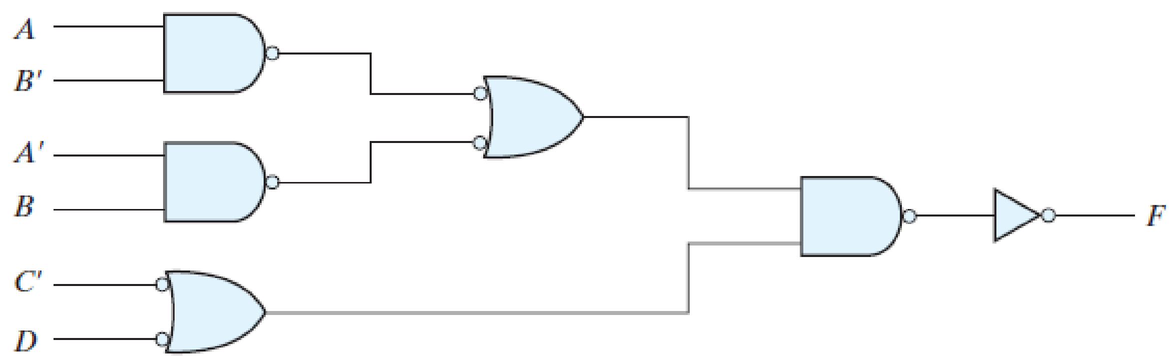
- Consider the multilevel Boolean function

$$F = (AB' + A'B)(C + D')$$

- Sol



(a) AND–OR gates

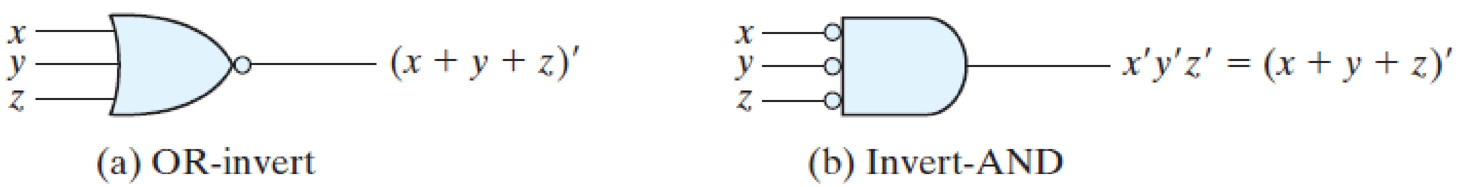


(b) NAND gates

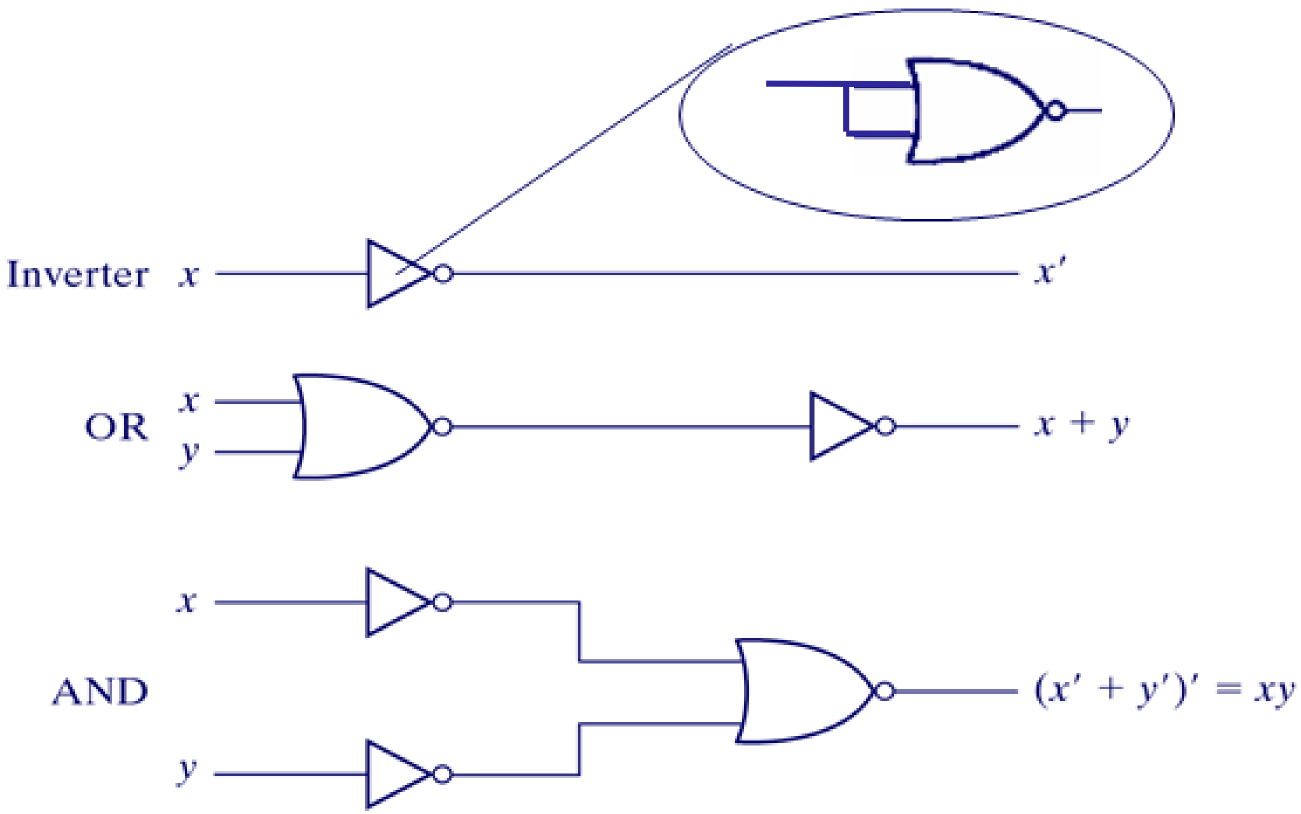
NOR Implementation

- The NOR operation is the dual of the NAND operation.
  - Procedures and rules for NOR logic are the duals NAND logic.

- **Universal gate** – any logic circuit can be implemented with it.



- Alternative graphic symbol for the NOR gate
  - In part (b), we can place a bubble (NOT) in each input and apply the DeMorgan's theorem, then get a Boolean function in NOR type.



Logic Operations with NOR Gates

Implementation

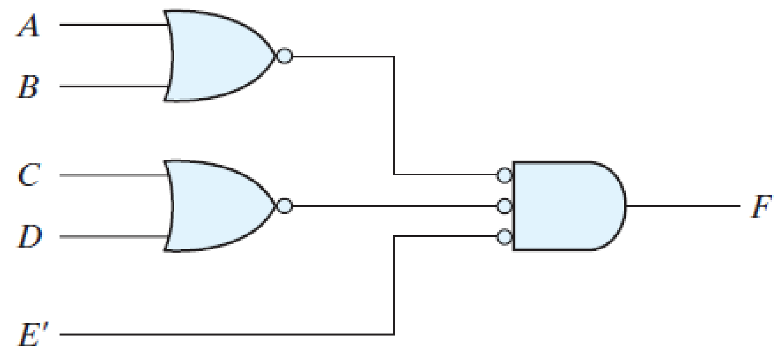
- Simplify the function into POS form.
  - Based on K-map - combining the 0's and complementing.

- Transformation from the OR–AND to a NOR diagram
  - Change the OR gates to NOR gates with OR-invert graphic symbols
  - Change the AND gate to a NOR gate with an invert-AND graphic symbol.
  - A single literal term going into the second-level gate must be complemented.

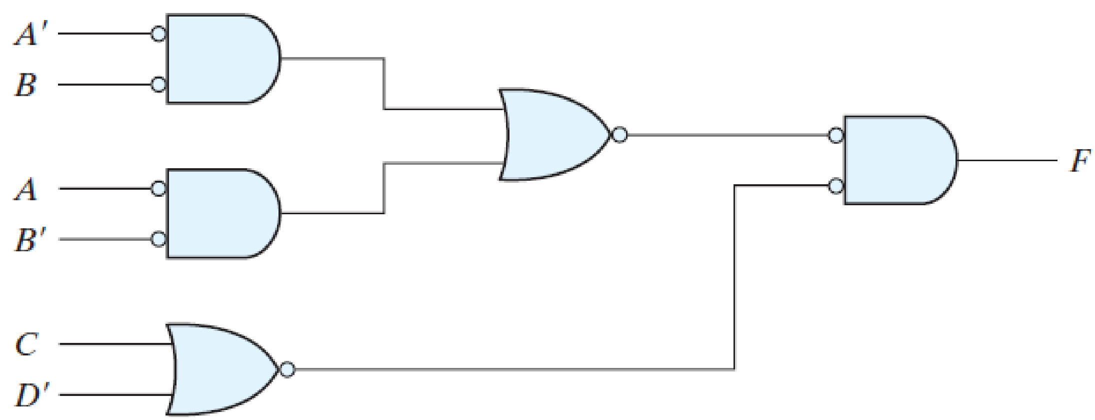
- Express the following function with a NOR diagram

$$F = (A + B)(C + D)E$$

$$F = (AB' + A'B)(C + D')$$



Implementing $F = (A + B)(C + D)E$



Implementing $F = (AB' + A'B)(C + D')$ with NOR gates

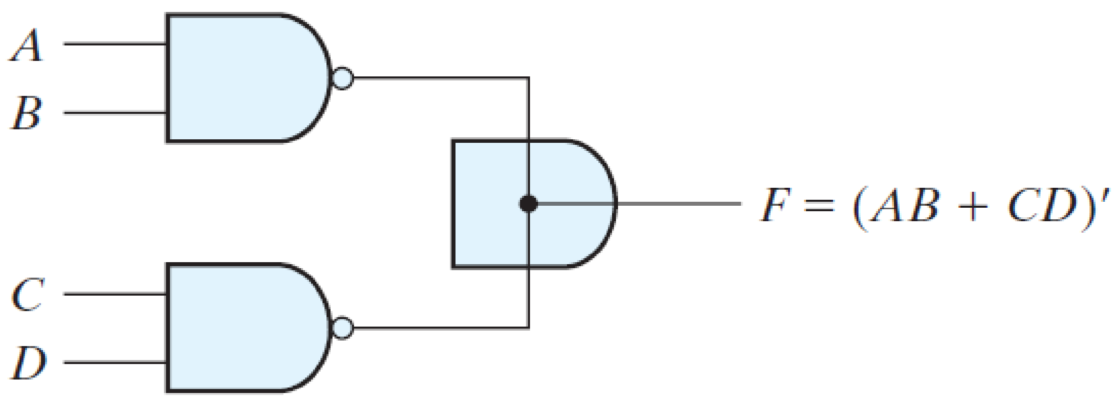
Other Two-level Implementations

- Some NAND or NOR gates allow the possibility of a wire connection between the outputs of two gates to provide a specific logic function.

- This type of logic is called *wired logic*.

- The logic function called an AND–OR–INVERT can be given as;

$$F = (AB)' \dots (CD)' = (AB + CD)' = (A' + B')(C' + D')$$

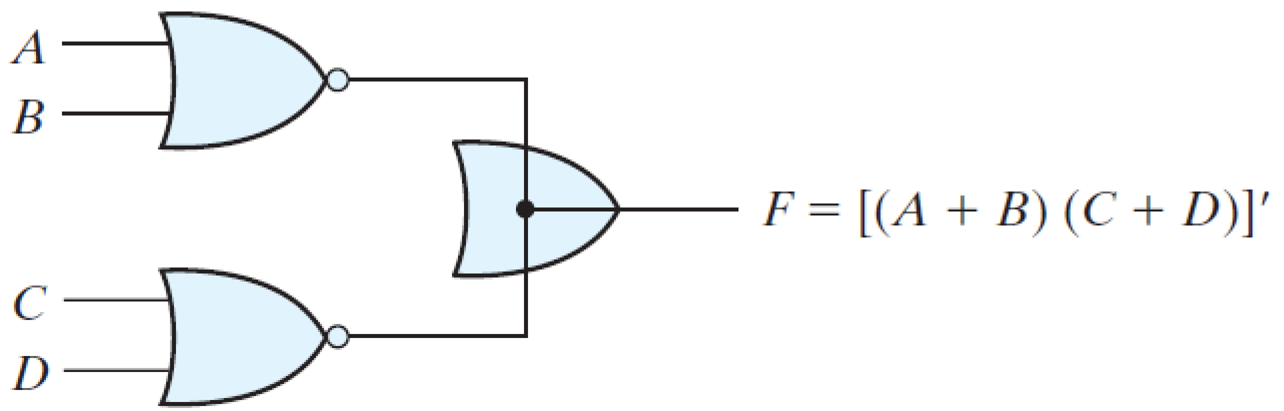


(a) Wired-AND in open-collector TTL NAND gates.

(AND–OR–INVERT)

- The NOR outputs of emitter-coupled logic (ECL) gates can be tied together to perform a wired-OR function called an OR–AND–INVERT function.

$$F = (A + B)' + (C + D)' = [(A + B)(C + D)]'$$



(b) Wired-OR in ECL gates

(OR–AND–INVERT)

- Some NAND or NOR gates allow the possibility of a wire connection between the outputs of two gates to provide a wired logic.
- Open-collector TTL NAND gates, when tied together, perform the wired-AND logic
- The wired-AND gate is not a physical gate.

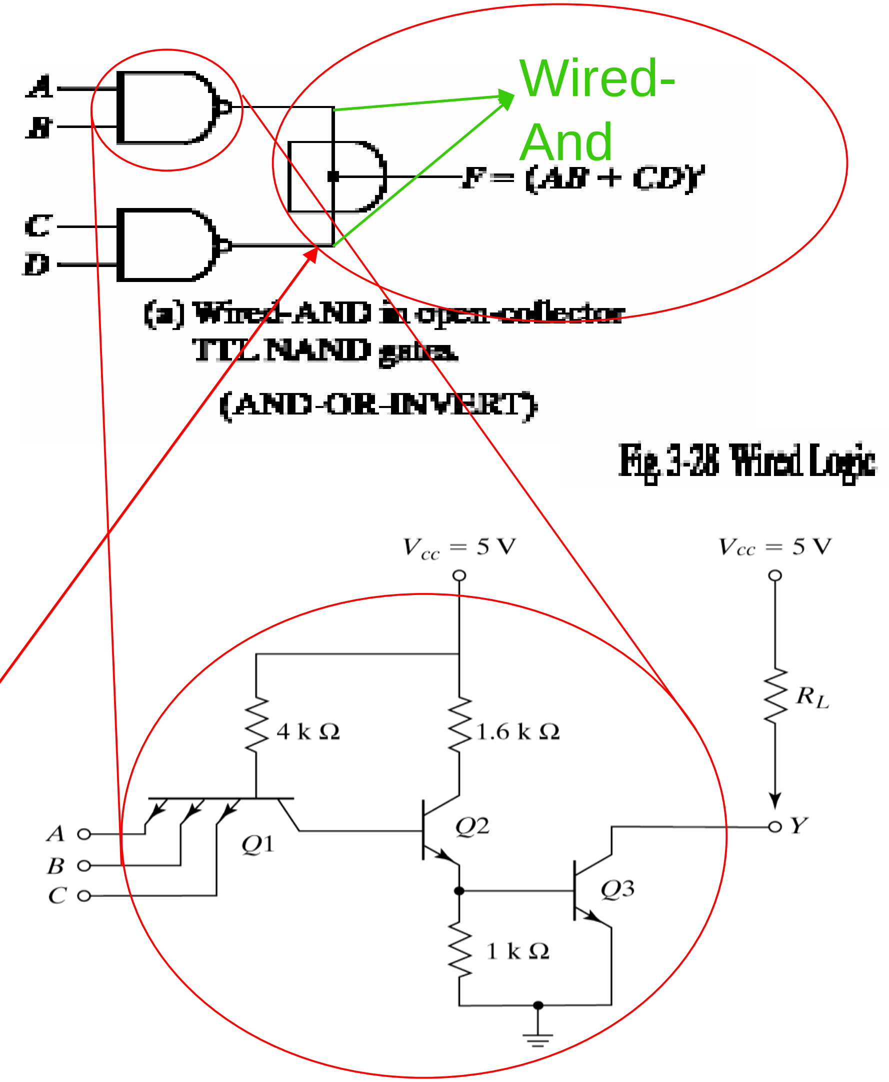


Fig. 10-11 Open-Collector TTL Gate

Nondegenerate forms

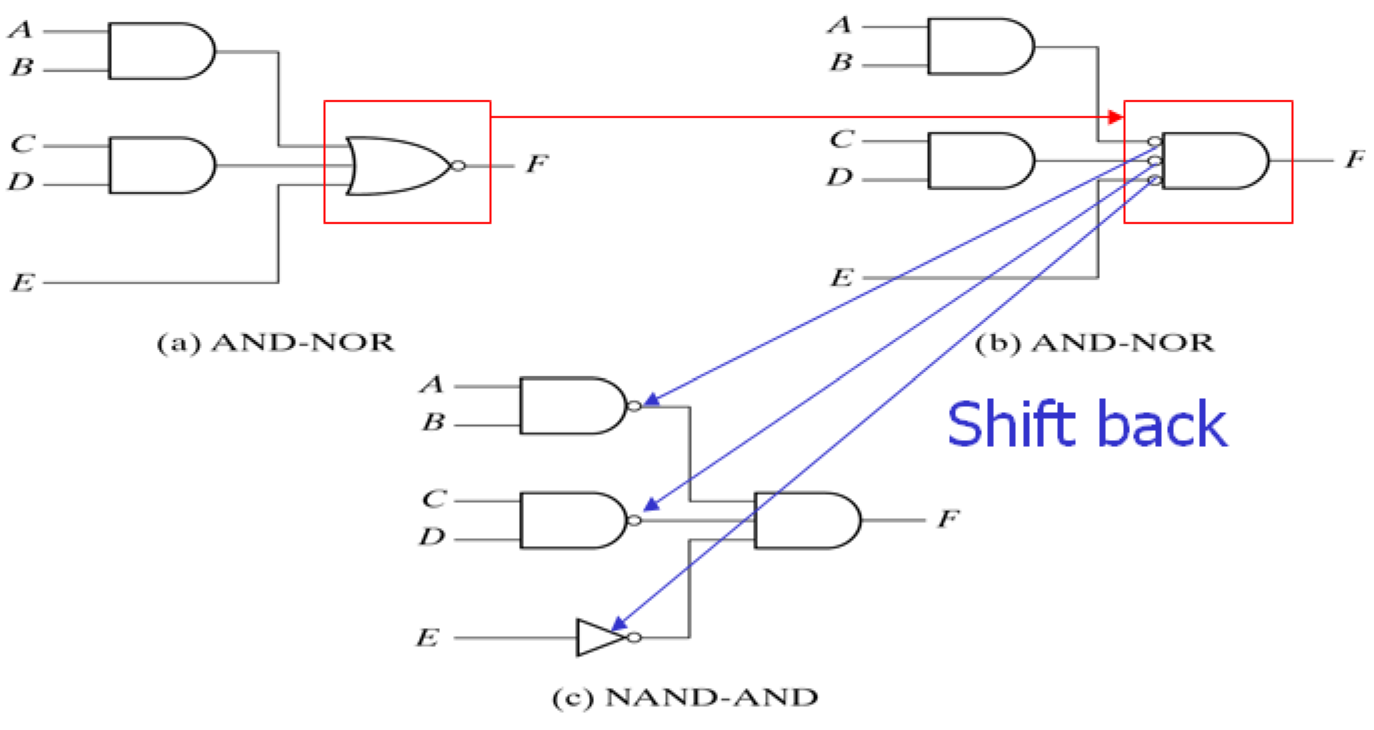
- We consider four types of gates: AND, OR, NAND, and NOR. These will have 16 combinations of two-level forms. *(by assigning one type of gate for the first level and one type for the second level)*
- Eight of these combinations are said to be degenerate forms, because they degenerate to a single operation.
- The other eight nondegenerate forms produce an SOPs or POSs as follows:

AND-OR → 3-4	OR-AND → 3-4
NAND-NAND → 3-6	NOR-NOR → 3-6
NOR-OR	NAND-AND
OR-NAND	AND-NOR

AND-OR-INVERT implementation

- The two forms NAND-AND and AND-NOR are equivalent forms and can be treated together.

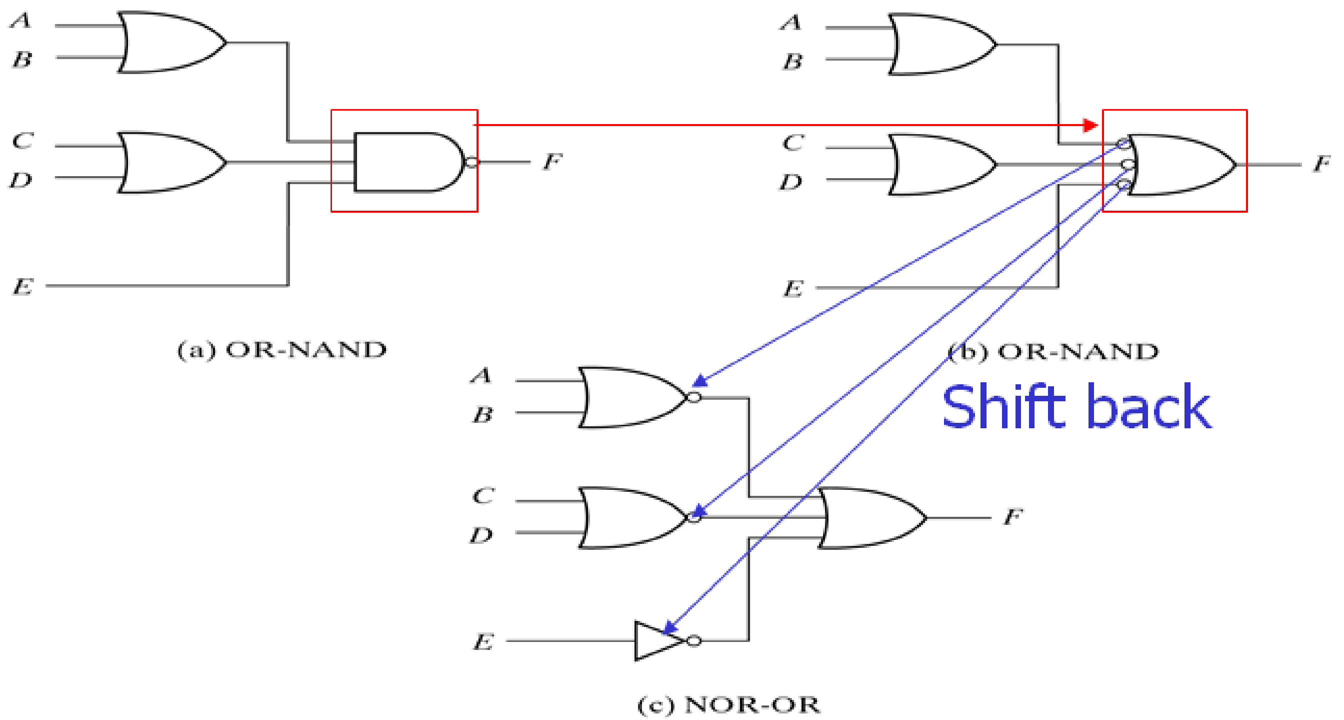
$$F = (AB + CD + E)'$$



AND-OR-INVERT Circuits; $F = (AB + CD + E)'$

OR-AND-INVERT implementation

- The OR–NAND and NOR–OR forms perform the OR–AND–INVERT function.
- The OR-NAND form resembles the OR-AND form, except for the inversion done by the bubble in the NAND gate.



OR-AND-INVERT Circuits; $F = [(A + B)(C + D)E]'$

Tabular summary

- Because of the INVERT part in each case, it is convenient to use the simplification of F' of the function.

Equivalent Nondegenerate Form (a)	Equivalent Nondegenerate Form (b)*	Implements the Function	Simplify F' into	To Get an Output of
AND–NOR	NAND–AND	AND–OR–INVERT	Sum-of-products form by combining 0's in the map.	F
OR–NAND	NOR–OR	OR–AND–INVERT	Product-of-sums form by combining 1's in the map and then complementing.	F

- Homework
  - Exclusive-OR Function
  - Odd Function
  - Parity Generation and Checking